



Объектная модель. Типы и структуры данных

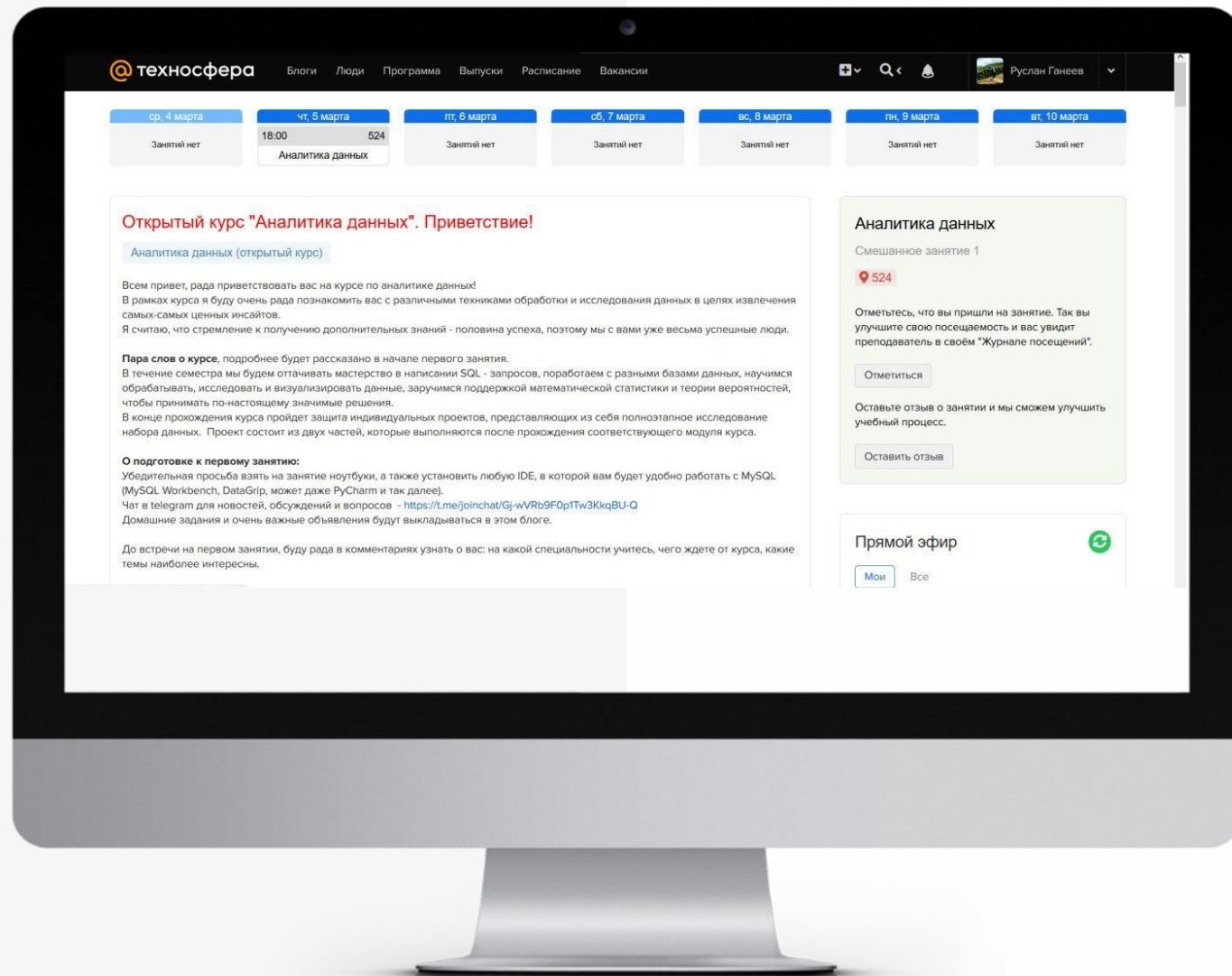
Антон Кухтичев





Содержание занятия

1. Стандартные типы в питоне
2. Магические поля объектов
3. Магические методы объектов
4. Методы кастомизации доступа к атрибутам
5. Методы кастомизации классов
6. Модуль collections:
defaultdict, OrderedDict, namedtuple,
deque, Counter



Напоминание отметиться на портале

Иначе плохо всё будет.




Всё есть объект

“Objects are Python’s abstraction for data. All data in a Python program is represented by objects or by relations between objects.”

docs.python.org



Объект

- 
1. Каждый объект имеет id, тип и значение
 2. Id никогда не меняется после создания объекта (is сравнивает id объектов)
 3. Тип объекта определяет какие операции с ним можно делать
 4. Значение объекта может меняться



Стандартные типы

Типы с одним значением

1. None
2. NotImplemented
3. Ellipsis (...)

Типы с одним значением

```
>>> None
>>> type(None)
<class 'NoneType'>
>>> NotImplemented
NotImplemented
>>> type(NotImplemented)
<class 'NotImplementedType'>
```

```
>>> ...
Ellipsis
>>> type(...)
<class 'ellipsis'>
>>> type(None)()
>>> type(None)() is None
True
>>> type(NotImplemented)() is
NotImplemented
True
```



Стандартные типы

numbers.Number

- numbers.Integral (int, bool)
- numbers.Real (float)
- numbers.Complex (complex)

numbers.Number

```
>>> import numbers
>>> issubclass(int, numbers.Number)
True
>>> issubclass(bool, int)
True
>>> issubclass(float, numbers.Real)
True
```

Стандартные типы

Sequences

Представляют собой конечные упорядоченные множества, которые проиндексированы неотрицательными числами.

Делятся на:

- `immutable` — Strings, Tuples, Bytes
- `mutable` — Lists, Byte Arrays



Стандартные типы

Set

Множество уникальных неизменяемых объектов. По множеству не индексируется, но по нему можно итерироваться

Существует 2 типа множеств:

- Sets
- Frozen sets



Стандартные типы

Mappings

Есть только 1 маппинг тип – `Dictionaries`. Ключами могут быть только неизменяемые типы, также стоит отметить, что `hash` от ключа должен выполняться за константное время, чтобы структура данных была эффективной.

Стандартные типы

Подумать

```
>>> a = {1.0}
```

```
>>> 1.0 in a
```

```
???
```

```
>>> 1 in a ???
```

```
>>> True in a
```

```
???
```



Стандартные типы

Модули

Модули являются основным компонентом организации кода в питоне (и это тоже объекты).

Стандартные типы

Callable types

- Пользовательские функции
- Методы класса
- Корутины
- Асинхронные генераторы
- Built-in methods
- Классы
- Экземпляры класса

Магические поля

Пользовательские функции

`__doc__` докстринг, изменяемое

`__name__` имя функции, изменяемое

`__qualname__` fully qualified имя, изменяемое

`__module__` имя модуля, в котором определена функция, изменяемое

Магические поля

Пользовательские функции

```
>>> def foo():
...     """aaaaaaa"""
...     pass
...
>>> foo.__doc__
'aaaaaaa'
>>> foo.__name__
'foo'
```

```
>>> def wrapper():
...     a= 1
...     def foo():
...         print(a)
...     return foo
...
>>> wrapper().__qualname__
'wrapper.<locals>.foo'
>>> wrapper.__module__
'__main__'
```

Магические поля

Пользовательские функции

`__defaults__` — tuple дефолтных значений, изменяемое

`__code__` — объект типа code, изменяемое

`__globals__` — словарь глобальных значений модуля, где функция объявлена, неизменяемое

`__dict__` — namespace функции, изменяемое

Магические поля

```
>>> def foo(a=1, b=2):  
...     pass  
...  
>>> foo.__defaults__  
(1, 2)  
>>> foo.__code__  
<code object foo at  
0x7f98fe73d660, file "<stdin>",  
line 1>
```

```
>>> foo.__globals__  
{... '__name__': '__main__',  
'numbers': <module 'numbers'  
from  
'/usr/local/lib/python3.7/number  
s.py'>...}  
>>> foo.a = 1  
>>> foo.__dict__  
{ 'a': 1 }
```



Магические поля

Пользовательские функции

`__annotations__` — словарь аннотаций, изменяемое

`__kwdefaults__` — словарь дефолтных значений кваргов, изменяемое

Магические поля

Пользовательские функции

```
>>> def foo(a: int, b: float):  
...     pass  
...  
>>> foo.__annotations__  
{'a': <class 'int'>, 'b': <class 'float'>}  
  >>> def foo(*, a=1, b=2):  
...     pass  
...  
>>> foo.__kwdefaults__  
{'a': 1, 'b': 2}
```



Магические поля

Пользовательские функции

`__closure__` — tuple ячеек, которые содержат биндинг к переменным замыкания

Магические поля

Классы

`__name__` — имя класса

`__module__` — модуль, в котором объявлен класс

`__qualname__` — fully qualified имя

`__doc__` — докстринг

`__annotations__` — аннотации статических полей класса

`__dict__` — namespace класса

Магические поля

`__self__` — объект класса

`__func__` — сама функция, которую мы в классе объявили

Магические поля

Методы

```
>>> class A:
...     def foo():
...         pass
...
>>> A.foo
<function A.foo at 0x1025929d8>
>>> A().foo
<bound method A.foo of <__main__.A object at 0x102595048>>
>>> A().foo.__func__
<function A.foo at 0x1025929d8>
>>> A().foo.__self__
<__main__.A object at 0x102595048>
```

Магические поля

Классы (поля, относящиеся к наследованию)

`__bases__` — базовые классы

`__base__` — базовый класс, который указан первым по порядку

`__mro__` — список классов, упорядоченный по вызову `super` функции

Магические поля

Классы (внутренности интерпретатора)

`__dictoffset__`

`__flags__`

`__itemsize__`

`__basicsize__`

`__weakrefoffset__`

Магические поля

Классы `__slots__`

Поле позволяет явно указать поля, которые будут в классе. В случае указания

`__slots__` пропадают поля `__dict__` и `__weakref__`

Используя `__slots__` можно сильно экономить на памяти и времени доступа к атрибутам объекта.

Магические поля

To string

`__repr__` — представление объекта. Если возможно должно быть валидное python выражение для создание такого же объекта

`__str__` — вызывается функциями `str`, `format`, `print`

`__format__` — вызывается при форматировании строки

Магические поля

Rich comparison

`object.__lt__(self, other)`

`object.__le__(self, other)`

`object.__eq__(self, other)`

`object.__ne__(self, other)`

`object.__gt__(self, other)`

`object.__ge__(self, other)`

`x < y == x.__lt__(y), <=, ==, !=, >, >=`

Магические поля

`__hash__`

Вызывается функцией `hash()` и коллекциями, которые построены на основе hash-таблиц. Нужно, чтобы у равных объектов был одинаковый hash

Если определен метод `__eq__` и не определен `__hash__`, то объект не может быть ключом в hashable коллекции. `__hash__` может быть определен только у неизменяемых типов

Магические поля

Эмуляция контейнеров

`object.__len__(self)`

`object.__length_hint__(self)`

`object.__getitem__(self, key)`

`object.__setitem__(self, key, value)`

`object.__delitem__(self, key)`

`object.__missing__(self, key)`

`object.__iter__(self)`

`object.__reversed__(self)`

`object.__contains__(self, item)`

Магические поля

Эмуляция контейнеров

```
object.__len__(self)
object.__length_hint__(self)
object.__getitem__(self, key)
object.__setitem__(self, key, value)
object.__delitem__(self, key)
object.__missing__(self, key)
object.__iter__(self)
object.__reversed__(self)
object.__contains__(self, item)
```

Магические поля

Эмуляция чисел

`object.__add__(self, other)`

`object.__sub__(self, other)`

`object.__mul__(self, other)`

`object.__matmul__(self, other)`

`object.__truediv__(self, other)`

`object.__floordiv__(self, other)`

`object.__mod__(self, other)`

`object.__divmod__(self, other)`

Магические поля

Эмуляция чисел

`object.__pow__(self, other[, modulo])`

`object.__lshift__(self, other)`

`object.__rshift__(self, other)`

`object.__and__(self, other)`

`object.__xor__(self, other)`

`object.__or__(self, other)`

Магические поля

Эмуляция чисел

Методы вызываются, когда выполняются операции (+, -, *, @, /, //, %, divmod(), pow(), **, <<, >>, &, ^, |) над объектами – `x + y == x.__add__(y)`

Есть все такие же с префиксом `r` и `i`.

`__radd__` - вызывается, если левый операнд не поддерживает `__add__`

`__iadd__` - вызывается, когда `x += y`

Магические поля

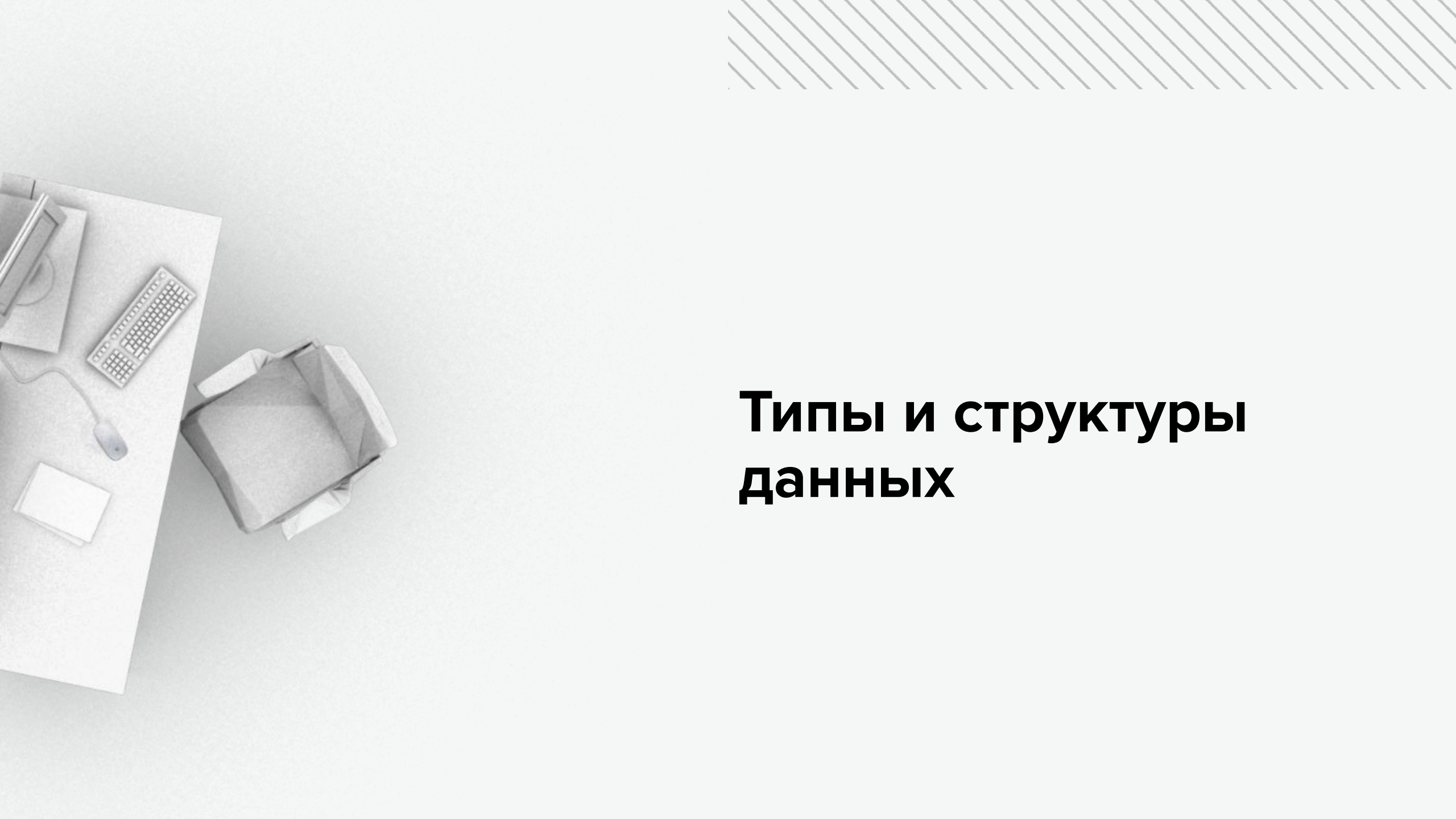
Эмуляция чисел

`object.__neg__(self)`

`object.__pos__(self)`

`object.__abs__(self)`


`object.__invert__(self)`



Типы и структуры данных



Что такое модуль `collections`?



Данный модуль реализует специализированные типы данных контейнера, предоставляя альтернативы для встроенных контейнеров таких как `dict`, `list`, `set` и `tuple`.

Что такое модуль collections?

Рассмотрим:

- `defaultdict`
- `OrderedDict`
- `namedtuple`
- `deque`
- `Counter`



defaultdict

`collections.defaultdict([default_factory[, ...]])`

Ничем не отличается от обычного словаря за исключением того, что по умолчанию всегда вызывается функция, возвращающая значение.

defaultdict | Пример

```
collections.defaultdict([default_factory[, ...]])
```

```
>>> import collections
```

```
>>> defdict = collections.defaultdict(list)
```

```
>>> print(defdict)
```

```
defaultdict(<class 'list'>, {})
```

```
>>> for i in range(5):
```

```
...     defdict[i].append(i)
```

```
...
```

```
>>> print(defdict)
```

```
>>> defaultdict(<class 'list'>, {0: [0], 1: [1], 2: [2], 3:  
[3], 4: [4]})
```



OrderedDict



```
collections.OrderedDict([items])
```

Похожий на словарь объект, но он помнит порядок, в котором ему были даны ключи.

OrderedDict | Пример

```
>>> import collections
>>> d = collections.OrderedDict(
...     [('a', 'A'), ('b', 'B'), ('c', 'C')]
... )
>>> for k, v in d.items():
...     print(k, v)
>>> d.move_to_end('b')
```



namedtuple



```
collections.namedtuple(typename, field_names, *, rename=False,  
defaults=None, module=None)
```

Именованные кортежи являются неизменяемыми подобно обычным кортежам.

Вы не можете изменять их после того, как вы что-то поместили в них.

namedtuple | Пример

```
>>> import collections
>>> Point = collections.namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)
>>> p[0] + p[1] # p = (11, 22)
33
>>> x, y = p
>>> x, y
(11, 22)
>>> p.x + p.y
33
```



namedtuple | Методы

- `_asdict()` - возвращает `OrderedDict` объекта
- `_make(iterable)`
- `_replace(**kwargs)`
- `_fields`
- `_fields_default`

namedtuple | Выводы

- `collections.namedtuple` — краткая форма для создания вручную эффективно работающего с памятью неизменяемого класса;
- Именованные кортежи могут помочь сделать ваш код чище, обеспечивая вас более простыми в понимании структурами данных;
- Именованные кортежи предоставляют несколько полезных вспомогательных методов которые начинаются с символа подчёркивания (`_`), но являются частью открытого интерфейса. Использовать их — это нормальная практика.

deque

```
collections.deque([iterable[, maxlen]])
```

Очередь из итерируемого объекта с максимальной длиной `maxlen`. Очереди очень похожи на списки, за исключением того, что добавлять и удалять элементы можно либо справа, либо слева.

deque | Пример

```
>>> from collections import deque
>>> d = deque('ghi') # make a new deque with three items
>>> d.append('j') # add a new entry to the right side
>>> d.appendleft('f') # add a new entry to the left side
>>> d # show the representation of the deque
deque(['f', 'g', 'h', 'i', 'j'])
>>> d.pop() # return and remove the rightmost item
'j'
>>> d.popleft() # return and remove the leftmost item
'f'
```

deque | Методы

- `append(x)/appendleft(x)`
- `clear()`
- `copy()`
- `count(x)`
- `extend(iterable)/extendleft(iterable)`
- `index(x[, start[, stop]])`
- `insert(i, x)`
- `pop()/popleft()`
- `remove(value)`
- `reverse()`



Counter



```
collections.Counter([iterable-or-mapping])
```

Это подкласс dict для подсчёта хешируемых объектов.

Counter | Методы

- `elements()`
- `most_common([n])`
- `subtract([iterable-or-mapping])`
- `update([iterable-or-mapping])`

Counter | Пример

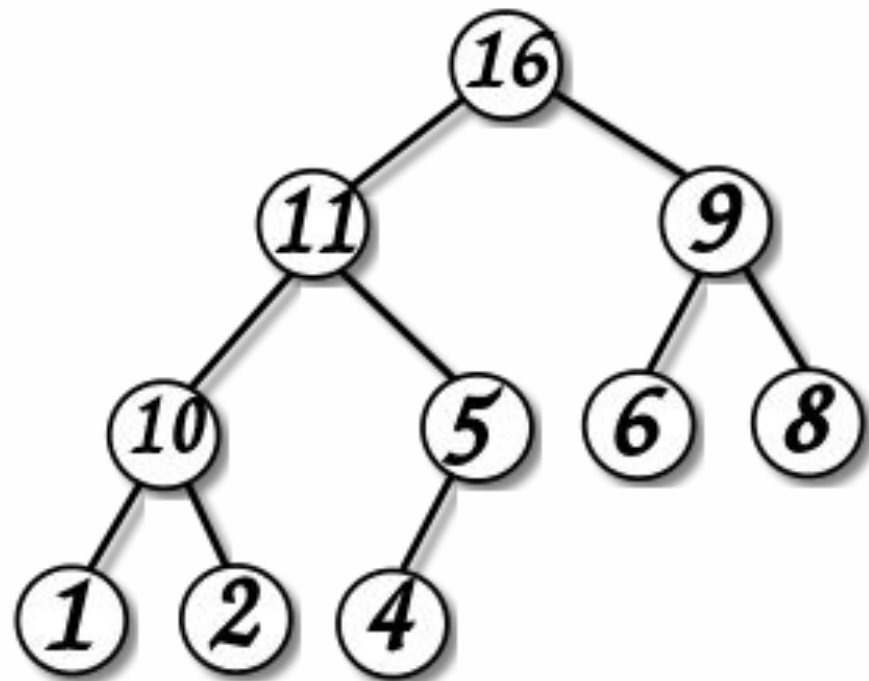
```
>>> import re
>>> words = re.findall(r'\w+',
                        open('hamlet.txt').read().lower())
>>> Counter(words).most_common(10)
[('the', 1143), ('and', 966), ('to', 762), ('of', 669),
 ('i', 631), ('you', 554), ('a', 546), ('my', 514), ('hamlet',
 471), ('in', 451)]
```

heap

Очередь с приоритетом. Реализована через кучу (heap).

1. Значение в любой вершине не меньше, чем значения её потомков;
2. Глубина всех листьев (расстояние до корня) отличается не более чем на 1 слой;
3. Последний слой заполняется слева направо без «дырок».

heapq



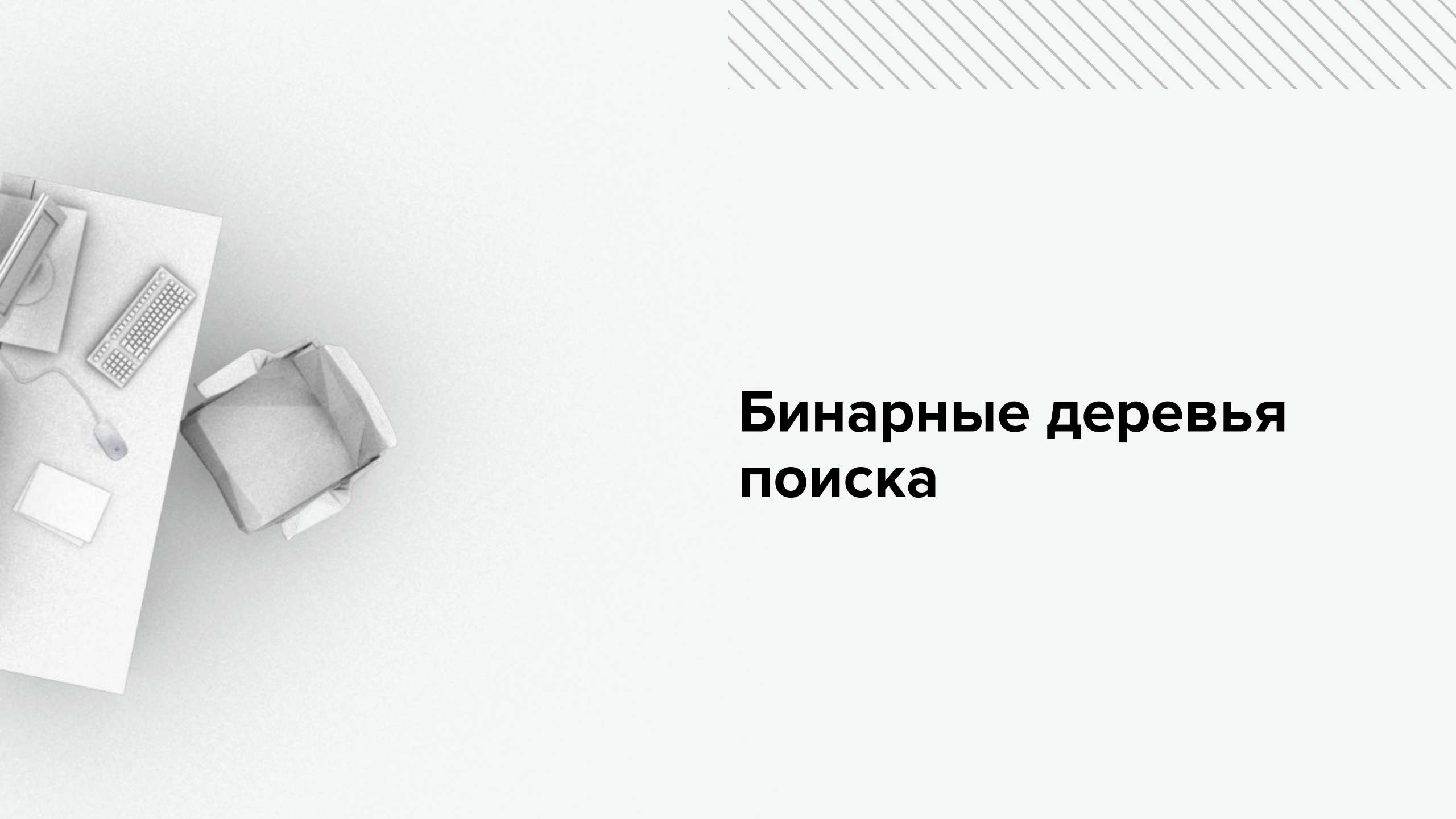
16	11	9	10	5	6	8	1	2	4
----	----	---	----	---	---	---	---	---	---

heapq | Методы

- `heapq.heappush(heap, item)`
- `heapq.heappop(heap)`
- `heapq.heappushpop(heap, item)`
- `heapq.heapify(x)`
- `heapq.heapreplace(heap, item)`
- `heapq.merge(*iterables)`
- `heapq.nlargest(n, iterable[, key])`
- `heapq.nsmallest(n, iterable[, key])`

heapq | Пример

```
>>> def heapsort(iterable):
...     h = []
...     for value in iterable:
...         heappush(h, value)
...     return [heappop(h) for i in range(len(h))]
...
>>> heapsort([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

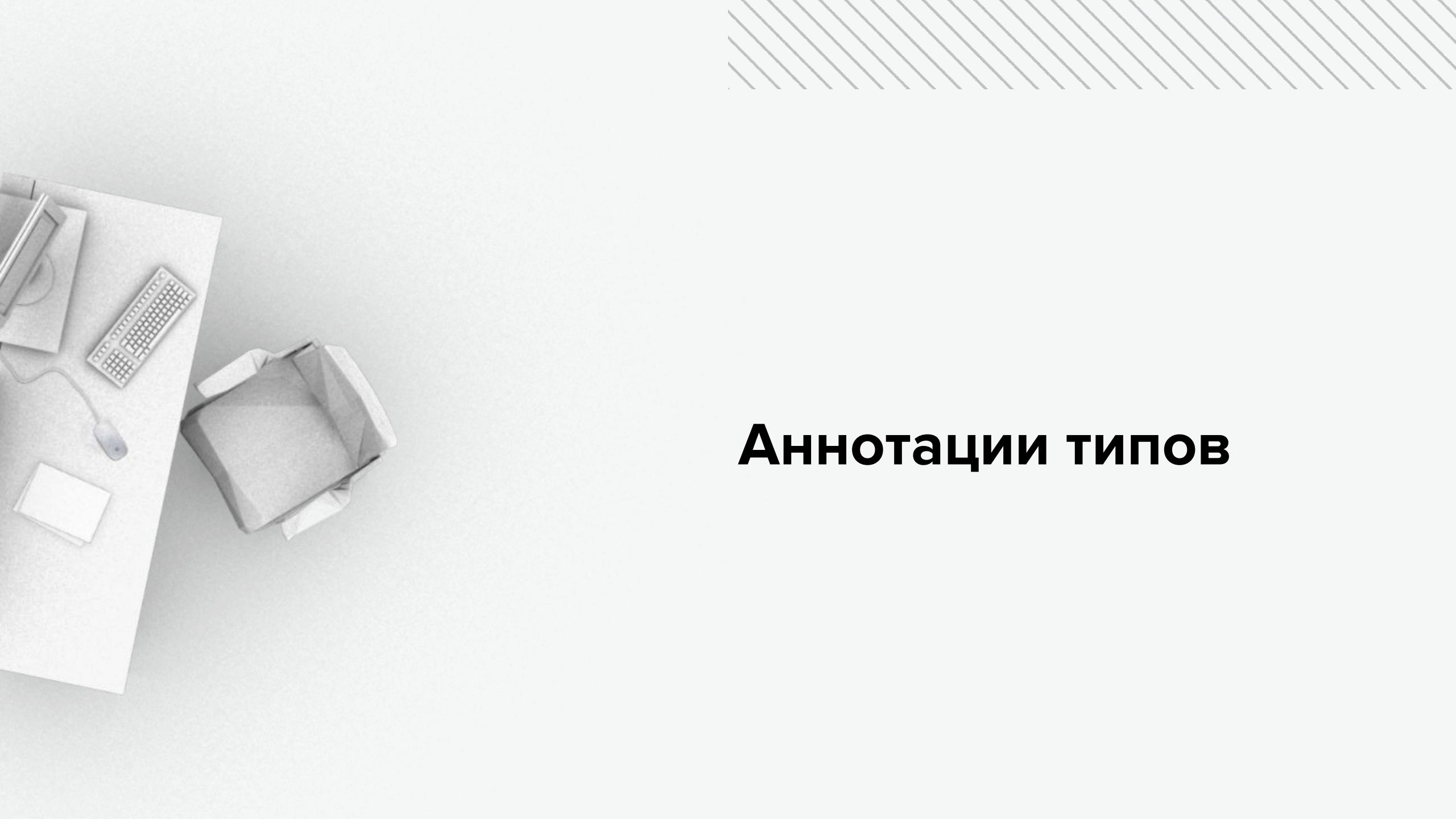


Бинарные деревья поиска

Бинарные деревья поиска. Binary search tree (BST)

БДП — это бинарное, или двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X .
- У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X .



Аннотации типов

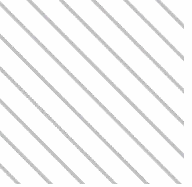
Аннотации типов

Аннотации типов просто считываются интерпретатором Python и никак более не обрабатываются.



Аннотации типов

- Пишутся непосредственно в коде;
- Повышается информативность исходного кода;
- Аннотации переменных:
 - `price: int = 5`
 - `title: str`
 - `titles: List[str] = ["hello", "world"]`
- Аннотации функций:
 - `def isBST(root: TreeNode) -> bool:`
 # Проверка, что БД является БДП.
 pass



Домашнее задание по уроку #2

Домашнее задание №2

#064

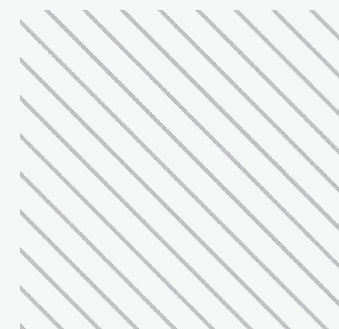
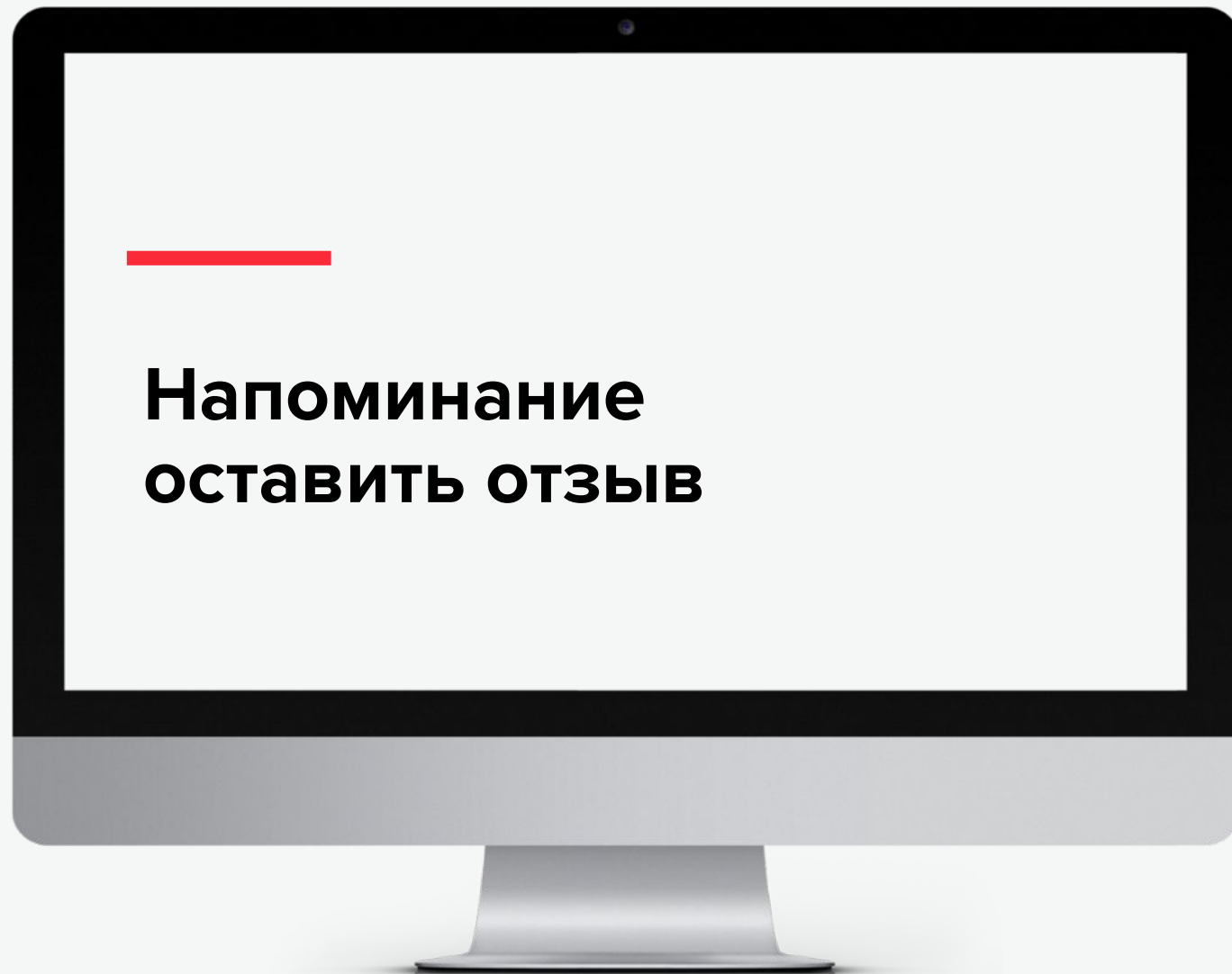
10

Баллов
за задание

13.10.20

Срок
сдачи

- Реализовать LRU cache (least recently used);
- Реализовать класс, отнаследованный от списка, такой, что один список.



**СПАСИБО
ЗА ВНИМАНИЕ**

